A General Framework for Information Processing with Application to Quantitative Software Testing

Vasile Rus and Sajjan Shiva
Department of Computer Science
The University of Memphis
Memphis, TN 38152
USA
{vrus, sshiva}@memphis.edu

Abstract

We propose in this paper a general framework for collecting and processing data from various sources of information. An advantage of our framework over traditional data mining frameworks is the ability to collect nuggets of information from sources encoded in different formats. We can handle documents or fragments of knowledge written in natural language (open structure documents), XML-like data (semi-structured documents), and database repositories (structured documents). Our proposed framework leads to an integrated technology for collecting data from these various sources as opposed to a technology in which there is a set of isolated components that deal with format-specific data sources.

We exemplify how this framework can help with the problem of finding duplicate defects. Before reporting a defect, a tester must check the defect has not been reported before. This checking step is time consuming because previous descriptions of same or similar defects may be worded differently, may appear in more than one repository (mailing lists, bug repository, known issues list), and may be encoded in different formats. Our information processing framework can provide an easy to use and efficient solution to this particular problem and other information seeking needs.

1. Introduction

Due to the complexity of modern software systems the information associated with the development and maintenance of software systems is ever increasing. Being able to offer fast and accurate access to information and to process the information to gain deeper insights about a system is a critical component of any modern software development effort. For instance, the Mozilla open source software project (http://www.mozilla.org) uses information systems that facilitate the acquisition of and easy access to information about various aspects of the project. Examples of repositories of information are developer’s forums (newsgroups and mailing lists), bug tracking system (Bugzilla), wikis, a query interface to the CVS source repository (Bonsai), crash reports and list of common crashes, browser to search through the source code, frequently reported bug lists. Most of the knowledge in these repositories is encoded in natural language. Searching in all these repositories requires advanced searching techniques based on natural language processing that go beyond simple keyword searching.

From a broader perspective, in today’s knowledge economy knowledge is becoming the new fourth factor of production besides labor, capital, and land. Knowledge acquisition and processing is nowadays a critical component of each organization. Putting together a generic list of risk factors or looking for documented defects related to a particular type of printer are two examples of daily tasks that every tester and test team must pay attention to. Such information seeking tasks, if aided by computers, can lead to significant productivity gains.

We describe a general framework for collecting, sharing, and using knowledge. We call the framework Systems Testing Expert System (STES). STES has two major components: (1) a repository of legacy knowledge that can be used to train and test employees on Systems Testing and (2) an Information Extraction tool for Systems Testing environments. We emphasize in this paper the latter component. An example of such a framework to identify duplicate bugs is also presented.

There are many advantages of STES. STES can lead to software testing process improvement and overall software development process improvement. The Information Extraction tool can be seen as a research vehicle and an important component of research infrastructure. It can be used to
enhance, validate, and compare data needed and produced by other Systems Testing research projects and by managers. STES can reveal caveats in the knowledge acquisition and knowledge transfer aspects of various projects, teams, and development and testing processes. We present here the general architecture of an Information Extraction tool for Systems Testing. The STES project can reduce the risk associated with various risk factors caused by lack of documentation, poor, or outdated information.

The benefit of an advanced information processing tool based on natural language processing is exemplified. We use the problem of finding duplicate bugs, called DUPS, as an example of a software testing task that can be significantly improved. Techniques based on word similarity can help better identify duplicate or similar bugs. Such word similarity measure can also help to cluster bugs which can then be used in various ways.

2 Related Work

Related work can be classified in two major categories. The first category refers to support systems for processing information related to software development projects. The second category includes efforts to use natural language processing for supporting software development. We elaborate on each category next.

Software Knowledge Bases (SKB) were systems built to track software components [7]. The main disadvantage of the early SKB was the isolated nature of the different tools/components. These SKB were not an integrated methodology. There were mailing lists, comments in version control system, comments in source code, requirements databases. This isolated architecture seems to be inherited even in today’s software development environments. Additionally, the SKB were not easy to use. The users, managers and developers, need to be trained in order to add or retrieve facts from such SKB. Modern knowledge bases are in more natural formats. Users can simply add facts or search for facts using natural language. Companies and open source projects are using wikis to record and retrieve knowledge about various aspects of software developments. Wikis are web portals in which the visitor of a page can edit the page using the browser. An example used by a company is Motoqwiki - http://www.motoqwiki.com - while an example of an wiki for an open source project is found at the following link: http://kb.mozillazine.org/Knowledge_Base. The use of wikis and other easy-to-use and efficient knowledge management technologies need to be implemented in systems testing. It will lead to increased productivity, reduced costs, reduced testing cycle time, and thus allowing more loads/releases in a calendar year.

The usage of natural language processing applications to improve software development and testing has been around for a while. More than a decade ago, Digital Equipment Corporation (DEC) used document parsers to automate software testing [6]. Even earlier, the CARPER project [11] aimed at generating a self-maintaining database checker automatically from fixed-format sections of documents describing a software project. Other projects were concerned with, for instance, parsing manuals to extract maximum allowed length of array of characters; or with automatically extracting software test conditions that would allow software testers to compare the behavior of a running system with the documentation describing its behavior [5]. The PATRicia system (Program Analysis Tool for Reuse) [3] was an information extraction system to automate understanding of object-oriented code as an aid to the reuse of object-oriented code. While this earlier application can be implemented and enhanced with modern techniques, there are new interesting information processing features that are enabled by deep natural language processing. We discuss next the use of Information Extraction as a general technology for extracting valuable facts from large and diverse repositories of information. The output of an Information Extraction system could be used in connection with machine learning techniques to automate the discovery of patterns and new knowledge related to testing.

An example is provided next. Let us pick the task of studying the relationship between defects and the time it takes to fix them, or the relationship between defects and delays in product releases. While an experienced developer may have an intuitive answer to the task, a quantitative model that would show evidence between various types of defects (failure, fault, error) and time to fix them would be ideal. The model could be used as a risk assessment tool and as a prediction tool for delays in releases or resources needs if on-time product release is a must. To build such a model the researcher would need to collect data on defects and the time it took to fix them. The data collection can be done manually by interviewing people, readings lots of documents, source codes, etc. Alternatively, the data can be automatically collected using an Information Extraction tool from comments in the code itself, from logs kept in the configuration management system, and other types of records. The use of an Information Extraction tool would greatly reduce the data collection time and data quality.

Thus, Information Extraction and Machine Learning are the two important components for Quantitative Software Testing, i.e. testing driven by quantitative analyzes of various aspects of the testing process.

2.1 What is Information Extraction?

Information Extraction is the science of automatically extracting facts from semi-structured and open structure documents, and arranging them in a structured, database-
like representation. Once in a structured representation (table-like) the data can be fed to machine learning algorithms (decision trees, nearest-neighbor) or statistical analysis tools (Excel, SPSS) to do regression and classification studies in order to discover hidden patterns and association rules that could improve the decision making process at project, team, division, and organization level. There has been a lot of effort to develop Information Extraction systems to extract facts from news articles (such as Wall Street Journal) in the financial and security (terrorism-related) domains [2, 1]. More recently there are Information Extraction systems for mining biomedical literature [4], and in the area of business intelligence, for instance to collect product-related opinions from blogs. The major task is to identify entities (such as person names, organization names, or gene names) and relations between those entities (Bill Gates CEO Microsoft would be an example of a CEO relationship between Bill Gates and Microsoft). The approaches to Information Extraction rely heavily on developing lexicosyntactic patterns to extract the desired information from textual fragments of documents (documents can be semi-structured or free text documents). The desired information is expressed as templates in the form of a set of fields. The role of the Information Extraction tool is to fill the templates with desired phrases extracted from the input. Existing Information Extraction techniques can be adapted to comment and identifier understanding, data mining from bug repository, mailing lists, and configuration management systems logs, etc. There are specific challenges in extracting information from software related repositories such as mailing lists, defect descriptions in bug\(^1\) repositories, tabular data, source code. For instance, comments of source code tend to be telegraphic, determiners being left out [3] as in Open file instead of Open the file, some comments are full sentences while others are not, some occur in a single line while others continue over several lines. Further, a team culture may have developed in the sense that some words have particular meanings for a team or even for a project. On the other hand, there are some peculiarities that make the task simpler than in a traditional setting of extracting information from news articles. That includes limited vocabulary as compared to the whole English vocabulary and limited syntactic structures such as comments being written mostly in present tense [3]. A comprehensive solution to extracting facts from free text descriptions of testing issues will have to address the above issues.

3 Motivation

There are at least two strong reasons that motivated this work.

\(^1\)We use in this paper the terms defect and bug interchangeable.
cess, we will create a repository of legacy knowledge that can be used as a resource for training and testing employees or job candidates.

STES can be useful for daily operations, in particular when crises appear and access to prior knowledge is timely needed. Among other possible usages, the tool could be adapted to generate tests from specification documents or to automatically check that design conforms with standards imposed by quality assurance bodies.

### 3.1 Broader Impacts

The STES system will benefit both industry and the academic world. For companies, STE can result in an improved process for knowledge acquisition, transfer, and retrieval in the area of testing with direct impact on software cycles, possibly facilitating more loads/releases per year, and deep cultural changes with direct impact on the overall operations. STES may reduce various risk factors which are caused by lack of documentation, poor, or outdated information. Additionally, it will reduce the training and testing costs with new and existing Systems Testing employees.

For academic purposes, STES creates opportunities to do research on both software testing and information retrieval areas.

STES can be applied to open source projects. These projects are developed by mixed teams of developers from different continents and cultures. This setup resembles the current practices of outsourcing in corporations where on-site employees work together with teams on other continents on common projects. By exploring the two worlds, open source versus corporate practices, valuable insights can be learned and transferred from one side to the other.

### 4 Proposed Architecture

The Architecture of the Systems Testing Expert System is presented in Figure 3. We do not show in this picture the collection of information component for pedagogical reasons. We focus on the extraction aspect of information processing. The reader should keep in mind that there are ways to populate the different sources of information with new entries. We briefly mentioned before the modern use of wikis for collecting and sharing information. There are more traditional ways that we do not elaborate on such as adding new entries to bug repositories or an expert compiling a document listing the known issues with a certain version of a software product. The Information Extraction tool will be able to mine the various sources of information (structured - databases, semi-structured - XML and source code, and free-text documents) for relevant facts and aggregate them in a smart presentation format as specified by the user. Examples of sources of information using different formats to encode data are shown in Figure 3. The presentation can be a nice report with text and tables, a summary, or a set of tuples that can be fed to a machine learning toolkit to do rigorous analysis and discover new patterns in the data. While querying for information in a database is somehow well defined, in our case there is one extra issue. We plan to allow the user the specify his queries as naturally as possible, i.e. using questions in natural language. For instance, a question/query might be *What are all the well known bugs related to printing problems reported in the last 48 hours?* Such a natural way of asking for information makes the user happier while challenging the developer to map the question into a database query. This mapping is the extra issue we mentioned above for querying databases. Retrieving relevant facts from the semi-structured and free text documents requires deeper natural language processing techniques such as Question Answering techniques [10].

### 5 An Example

We now exemplify how an advanced information extraction tool based on deeper language processing techniques is better than a simple keyword search tool that is currently available in the various interfaces to information repositories.

One of the important steps of reporting a bug is to make sure it has not been previously reported. According to the Mozilla tutorial on screening duplicate bugs “probably less than half turn out to be duplicates, it is best to presume that each one is until you convince yourself otherwise”. The guidelines for finding duplicates sends the tester to Mozilla QA - Most Frequent Bugs List, the Known Issues section of the Firefox release notes or the release notes of the latest Mozilla suite version, the list of bugs recently reported by the Smoketesting team, and the netscape.public.mozilla.builds newsgroup. For instance, the Most Frequent Bugs List offers thousands of entries. A simple search feature based on keywords is available but offers results biased towards the exact keywords used in the query, lacking deeper search capabilities that natural language based techniques could offer. Such shallow technique will miss critical bugs as shown below.

The above sources of bug information have for each bug report/entry a field the describes the bug in details. The description is written in natural language. Our proposed solution is to use this description in conjunction with word similarity measures [8, 9] to find similarities among bugs and between bugs and the user query. Further, it is important to have a powerful interface that would allow testers to enter few keywords and then return entries from all sources instead of testers having to do individual searches on every resource.

Let us discuss an example that would illustrate the need
Software Testing is the process used to help identify correctness, completeness, security, and quality of developed software. Testing includes executing a program to find errors.

Figure 1. Architecture of an Information Processing System for Systems Testing.

<table>
<thead>
<tr>
<th>ID</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>341200</td>
<td>normal</td>
<td>can’t use full screen area after 1.5.0.4 upgrade on Mac OS 10.4 with 23 inch display (Firefox window keeps bouncing to center of desktop)</td>
</tr>
<tr>
<td>313029</td>
<td>critical</td>
<td>Display window bounces once tabbed browsing is enabled.</td>
</tr>
</tbody>
</table>

Table 1. Two similar bugs from the Mozilla’s Most Frequently Reported Bugs list.
Figure 2. Sources of Defect Information in the Mozilla Open Source Project
for deeper natural language processing. The Known Issue section of version 2.0.0.2 of Mozilla Firefox lists windows either bouncing or shaking when no items are in the toolbar and in other cases as one of the known issues. If someone wants to report a bug on windows bouncing one can use the query windows bouncing to search for identical or similar bugs in the Most Frequently Reported Bugs list. If such a query is used a single bug (id#341200) is found with severity normal. Its description is can’t use full screen area after 1.5.0.4 upgrade on Mac OS 10.4 with 23 inch display (Firefox window keeps bouncing to center of desktop). However, a slightly different query Window bounces would miss the previous normal bug and reveal a new, critical bug. The new critical bug is bug id#313029 having description Display window bounces once tabbed browsing is enabled. Table 1 lists the two bugs. A search mechanism that relies on morphological processing of words should be able to retrieve both bugs. Lemmatization is the process of mapping all morphological variations of a word to its lemma or root form. As an example, go, went, gone, and going would be mapped to the root form go. Using such a mechanism words in the query and the repository can be reduced to their lemma before the search begins. The query Windows bouncing would thus become window bounce and the query Window bounces would become window bounce. The two slightly different queries become identical which is good because they have same meaning. Lemmatization based approaches can solve the problem of morphological variations of words used in queries. Lemmatization can not solve problems with queries that use different words with same meaning to express same information need. Deeper approaches are possible such as word similarity based approaches. For instance, a search mechanism based on word similarity techniques should be able to retrieve not only the two previous bugs but also similar bugs that are described with similar but not identical lemmas. Further, word similarity based techniques could facilitate clustering of bugs.

6 Conclusions

We presented in this paper a general framework for processing information in the area of software testing. We exemplified how the proposed framework can help with finding duplicate bugs in repositories of information about a software project. We plan to extend this work to collect defect information about software modules and then use that information to build defect predictors.

ACKNOWLEDGEMENT

This work was sponsored by The University of Memphis under a Systems Testing Excellence Program (STEP) project. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of The University of Memphis or the STEP program.

References